

# From Node Embedding To Community Embedding

Vincent W. Zheng<sup>1</sup>, Sandro Cavallari<sup>2</sup>, Hongyun Cai<sup>1</sup>, Kevin Chen-Chuan Chang<sup>3</sup>, Erik Cambria<sup>2</sup>

<sup>1</sup> Advanced Digital Sciences Center, Singapore;

<sup>2</sup> Nanyang Technological University, Singapore;

<sup>3</sup> University of Illinois at Urbana-Champaign, USA

{vincent.zheng, hongyun.c}@adsc.com.sg, sandro001@e.ntu.edu.sg, kcchang@illinois.edu, cambria@ntu.edu.sg

## Abstract

Most of the existing graph embedding methods focus on nodes, which aim to output a vector representation for each node in the graph such that two nodes being “close” on the graph are close too in the low-dimensional space. Despite the success of embedding individual nodes for graph analytics, we notice that an important concept of embedding communities (*i.e.*, groups of nodes) is missing. Embedding communities is useful, not only for supporting various community-level applications, but also to help preserve community structure in graph embedding. In fact, we see community embedding as providing a higher-order proximity to define the node closeness, whereas most of the popular graph embedding methods focus on first-order and/or second-order proximities. To learn the community embedding, we hinge upon the insight that community embedding and node embedding reinforce with each other. As a result, we propose ComEmbed, the first community embedding method, which jointly optimizes the community embedding and node embedding together. We evaluate ComEmbed on real-world data sets. We show it outperforms the state-of-the-art baselines in both tasks of node classification and community prediction.

## Introduction

Graph data is becoming increasingly popular, thanks to the proliferation of various social media (*e.g.*, blogs, Flickr, Twitter) and many other kinds of information networks (*e.g.*, DBLP, knowledge graphs). To effectively process and analyze the graph data, we often need to consider how to appropriately represent the graph. Graph embedding is a mainstream graph representation framework (Roweis and Saul 2000; Chang et al. 2015; Niepert, Ahmed, and Kutzkov 2016; Xie et al. 2016), which aims to project a graph into a low-dimensional space for further analytic tasks, such as classification, clustering and so on.

Conventionally, graph embedding focuses on *nodes* – it tries to output a vector representation for each node in the graph, such that two nodes being “close” on the graph have similar vector representations (*i.e.*, close in the low-dimensional space). There are different ways to measure the closeness between two nodes in the graph. Most of the existing graph embedding methods measure the closeness by:

1) *first-order proximity* (Tenenbaum, de Silva, and Langford 2000; Belkin and Niyogi 2001), which considers two nodes as close if they are direct neighbors to each other in the graph; 2) *second-order proximity* (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016), which considers two nodes as close if they share similar neighbors in the graph; 3) combination of first-order proximity and second-order proximity (Tang et al. 2015; Wang, Cui, and Zhu 2016), which considers two nodes as close if they are directly linked and also share similar neighbors.

Despite the success of embedding individual nodes for graph analytics, we notice that an important concept of embedding *communities* (*i.e.*, groups of nodes) is missing in the literature. Embedding community is useful; it helps to

- Support community-level applications: for example, we can visualize the communities in a low-dimensional space to help generate insights about the graph structure. We can also enable community recommendation by predicting the most likely community to a node according to their closeness in the embedded space. We will quantitatively evaluate this community prediction task in the experiment.
- Preserve community structure in graph embedding: the state of the art such as DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015) and node2vec (Grover and Leskovec 2016) is unable to preserve communities in the embedded space. In Fig. 1, we visualize the graph embedding of these methods for the Zachary’s Karate Club data set<sup>1</sup>. The graph has 34 nodes. Each node has a color<sup>2</sup>, and there are in total four colors, indicating four different communities. Fig. 1(b)–1(d) show that, they cannot clearly separate the nodes of different colors apart.

**Community embedding.** In this paper, we introduce a new concept of *community embedding*. A community embedding is a latent representation for a community in the graph. To represent a community, we are motivated by the Gaussian Mixture Model (Bishop 2006) to see each community as a Gaussian component. Thus, we represent each community with: 1) where its center is; 2) how its member nodes are spreaded. As to be defined later in Def. 1, we formulate a community embedding as a tuple of a mean vector indicat-

<sup>1</sup><https://networkdata.ics.uci.edu/data.php?id=105>

<sup>2</sup>Same colors as used in (Perozzi, Al-Rfou, and Skiena 2014).

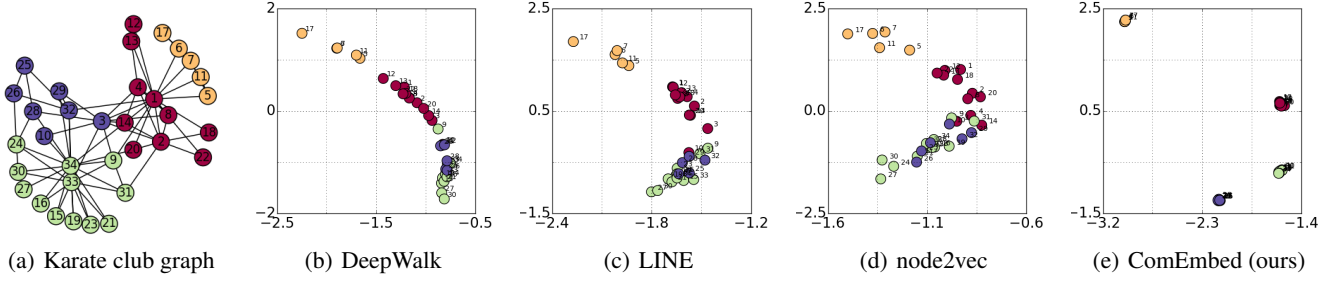


Figure 1: Effect of community embedding on the Zachary’s Karate Club data set. We embed the graph into a 2D space. We use the same data input for all the methods. For each node, we sample 10 paths, each of length 80. We set the context window size as 3 and negative sampling size as 5. In LINE, we use mean pooling, instead of concatenation, of two embeddings learned from first and second order proximity. In node2vec, we set  $p = 0.25$ ,  $q = 0.25$ . In ComEmbed, we set  $\alpha = 0.1$ ,  $\beta = 0.01$ .

ing the center of a community and a covariance matrix indicating the spread of its members in a low-dimensional space.

Community embedding is significant. First of all, it provides a “direct” representation for each community in the low-dimensional space, in contrast with representing a community as a collection of nodes. Second, it captures a *higher-order* proximity in graph embedding, such that the multi-hop-away nodes within the same community can also be close in the embedded space. There is few graph embedding work that considers higher-order proximity, and their definitions of higher-order proximity are not based on communities (Cao, Lu, and Xu 2016; Ou et al. 2016). More discussions are in the related work section.

**Our solution.** Our major insight to learn the community embedding is hinged upon the mutual reinforcement between node embedding and community embedding. On one hand, a good community embedding helps to get a good node embedding, because of the higher-order proximity. On the other hand, a good node embedding also helps to get a good community embedding, since the nodes can then be better clustered together. As a result, we shall jointly optimize the community embedding and the node embedding. Note that, having a better node embedding also enables us to do better predictions (*e.g.*, node classification) on the graph. We will also evaluate the node classification task in the experiment.

Based on our above insight, we propose ComEmbed, the first community embedding method for graph analytics. Specifically, in ComEmbed we iteratively optimize the community embedding and the node embedding. Given the node embedding, we fit the community embedding by the Gaussian Mixture Model. Given the community embedding, we fit the node embedding by novelly preserving all of the first-order, second-order and higher-order proximities. We emphasize that, the idea of joint modeling node embedding and community embedding is new. There is some work on using community to improve the node embedding (Yang et al. 2016), or using node embedding to achieve better community detection (Tian et al. 2014). But there is no study on the reinforcement of node embedding and community embedding. Besides, the joint modeling is also effective. In Figure 1(e), we first show some promising result: thanks to the com-

munity embedding, ComEmbed can well separate the nodes with different colors in the embedded space. More evaluations are in the experiment section.

We summarize our contributions as follows.

- To the best of our knowledge, we are the first to introduce the concept of community embedding to graph analytics.
- We propose ComEmbed, which jointly learns the community embedding and the node embedding together.
- We evaluate ComEmbed on real-world data sets. We improve the state-of-the-art baselines by at least 4.0%–5.5% (conductance) and 5.3%–11.2% (NMI) in community prediction, 14.1%–91.8% (macro-F1) and 7.6%–10.2% (micro-F1) in node classification.

## Related Work

There is no existing graph embedding work that considers community embedding. Most of the graph embedding methods focus on generating node embedding. For example, earlier methods, such as MDS (Cox and Cox 2000), LLE (Roweis and Saul 2000), IsoMap (Tenenbaum, de Silva, and Langford 2000) and Laplacian eigenmap (Belkin and Niyogi 2001), typically aim to solve the leading eigenvectors of graph affinity matrices as node embedding. Recent methods typically rely on neural networks to learn the representation for each node, with either shallow architectures (Yang, Cohen, and Salakhutdinov 2016; Xie et al. 2016; Perozzi, Al-Rfou, and Skiena 2014; Tang et al. 2015; Grover and Leskovec 2016) or deep architectures (Niepert, Ahmed, and Kutzkov 2016; Wang, Cui, and Zhu 2016; Chang et al. 2015). Other than node embedding, there is some attempt to learn edge embedding (Luo et al. 2015), which aims to learn the embedding for both entities (*i.e.*, nodes) and relations (*i.e.*, edges) in a knowledge graph.

There is little existing graph embedding work that considers high-order proximity and their definitions of high-order proximity are also different from ours. Most existing methods focus on first-order proximity and/or second-order proximity (Perozzi, Al-Rfou, and Skiena 2014; Grover and Leskovec 2016; Tang et al. 2015; Wang, Cui, and Zhu 2016), as discussed in the introduction. Some recent attempts to consider higher-order proximity are based on the facts that:

1) DeepWalk is a realization of SkipGram (Mikolov et al. 2013) on the graph data, where DeepWalk treats each node as a “word” and each path as a “sentence”; 2) SkipGram implicitly factorizes a matrix based on the word-word cooccurrence (Levy and Goldberg 2014). Thus, DeepWalk’s second-order proximity stems from factorizing the matrix about node-node cooccurrence. Both (Ou et al. 2016) and (Cao, Lu, and Xu 2016) design and factorize a higher-order node-node proximity matrix by PageRank or Katz index. Thus their higher-order proximity is based on the graph reachability via random walk, where the notion of community is missing.

The mutual reinforcement of community embedding and node embedding has never been exploited. It is common to use node embedding results for community detection (Tian et al. 2014; Kozdoba and Mannor 2015), but they do not have the notion of community in their node embedding. There is some work that allows community feedback to guide the node embedding (Yang et al. 2016), but again it lacks the concept of community embedding and its community feedback requires extra supervision on must-links.

### Problem Formulation

As *input*, we are given a graph  $G = (V, E)$ , where  $V$  is the node set and  $E$  is the edge set. Traditional graph embedding aims to learn a node embedding for each  $v_i \in V$  as  $\phi_i \in \mathbb{R}^d$ . In this paper, we introduce the concept of community embedding. Suppose there are  $K$  communities on the graph  $G$ . For each node  $v_i$ , we denote its community assignment as  $z_i \in \{1, \dots, K\}$ . Motivated by Gaussian Mixture Model (GMM), we represent each community as a Gaussian component, which is characterized by a mean vector indicating the community center and a covariance matrix indicating its member nodes’ spread. Formally, we define:

**Definition 1 (Community Embedding)** A community embedding for a community  $k$  in a  $d$ -dimensional space is a tuple  $(\psi_k, \Sigma_k)$ , where  $\psi_k \in \mathbb{R}^d$  is a Gaussian mean vector and  $\Sigma_k \in \mathbb{R}^{d \times d}$  is a Gaussian covariance matrix.

As *output*, we aim to learn both the community embedding  $(\psi_k, \Sigma_k)$  for each community  $k \in \{1, \dots, K\}$  and the node embedding  $\phi_i$  for each node  $v_i \in V$  on the graph  $G$ .

Considering the mutual reinforcement between node embedding and community embedding, we propose to learn two embeddings together. First of all, to learn the node embedding, we consider both first-order and second-order proximity. In addition, we leverage the community embedding to achieve higher-order proximity, by requiring all the nodes within the same community to share similar embeddings. More precisely, we require all the nodes to have their node embeddings “close” to the corresponding community embedding, where we measure the closeness by a Gaussian distribution. Finally, we jointly optimize the node embedding and the community embedding, by maximizing the likelihood of using both embeddings to preserve all the first-order, second-order and higher-order proximities on the graph. Next, we give the model details.

**First-order proximity.** To preserve the first-order proximity, we require two nodes that are direct neighbors on the

graph to have similar node embeddings. Specifically, for each edge  $(v_i, v_j) \in E$ , we follow LINE (Tang et al. 2015) to model the likelihood of first-order proximity as

$$p_1(v_i, v_j) = \sigma(\phi_j^T \phi_i), \quad (1)$$

where  $\sigma(x) = 1/(1 + \exp(-x))$  is a sigmoid function. Then we define the objective function for the first-order proximity node embedding as

$$O_1 = -\sum_{(v_i, v_j) \in E} \log p_1(v_i, v_j). \quad (2)$$

By minimizing  $O_1$ , we make  $\phi_i$  close to each of its direct neighbors  $\phi_j, \forall (v_i, v_j) \in E$ .

**Second-order proximity.** To preserve the second-order proximity, we require two nodes that share similar neighbors on the graph to have similar node embeddings. We follow DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) to consider a general sense of “neighbors” as the nodes that are reachable by the target node within  $\zeta$  steps in a random walk on the graph. Formally, we consider the neighbors of  $v_i$  as the *context* for  $v_i$ , and we denote  $C(v_i)$  as the context nodes for  $v_i$ . Then, to preserve the second-order proximity, we require node  $v_i$  to have similar node embedding to each of its context  $u_j \in C(v_i)$ . We follow LINE to introduce an extra *context embedding*  $\phi'_j \in \mathbb{R}^d$  for each node  $u_j$ . Thus we define the likelihood of second-order proximity as

$$p(C(v_i)|v_i; \phi, \phi') = \prod_{u_j \in C(v_i)} p(u_j|v_i; \phi, \phi'), \quad (3)$$

where  $p(u_j|v_i; \phi, \phi')$  in general is a softmax function

$$p(u_j|v_i; \phi, \phi') = \frac{\exp(\phi_j'^T \phi_i)}{\sum_{u_k} \exp(\phi_k'^T \phi_i)}.$$

As the summarization in the softmax function is time consuming, we follow (Mikolov et al. 2013) to use *negative sampling* to replace the summation term in the softmax function. By taking the logarithm over  $p(u_j|v_i; \phi, \phi')$ , we then define the new log-likelihood with negative sampling as

$$\begin{aligned} \log p_2(u_j|v_i; \phi, \phi') &= \log \sigma(\phi_j'^T \phi_i) \\ &+ \sum_{l=1}^m \mathbb{E}_{u_k \sim P_n(u)} [\log \sigma(-\phi_k'^T \phi_i)], \end{aligned} \quad (4)$$

where  $u_k \sim P_n(u)$  means sampling a node  $u_k$  (other than  $v_i$  and any of its context) from  $V$  as a *negative context* of  $v_i$  according to a probability  $P_n(u)$ . We follow (Tang et al. 2015) to define  $P_n(u) \propto r_u^{3/4}$ , where  $r_u$  is node  $u$ ’s degree.  $\mathbb{E}_{u_k \sim P_n(u)}[\cdot]$  is the expectation over  $P_n(u)$ . In total, we sample  $m$  negative context nodes to evaluate  $p(u_j|v_i; \phi, \phi')$ . Finally, we define the objective function for the second-order proximity node embedding as

$$O_2 = -\alpha \sum_{v_i \in V} \sum_{u_j \in C(v_i)} \log p_2(u_j|v_i; \phi, \phi'), \quad (5)$$

where  $\alpha > 0$  is a trade-off parameter. By minimizing  $O_2$ , we make  $\phi_i$  close to each of its context  $\phi_j, \forall \phi_j \in C(v_i)$ .

**Community embedding.** To preserve the higher-order proximity, we require all the nodes within the same community to be close to the corresponding community center in the embedded space. In other words, the community

structure is preserved after embedding the nodes into a low-dimensional space. To model the node distribution for each community in the embedded space, we choose GMM to model the likelihood of higher-order proximity as

$$\prod_{i=1}^N \sum_{k=1}^K p(v_i | z_i = k; \phi, \psi, \Sigma_k) p(z_i = k), \quad (6)$$

where each node  $v_i$  belongs to a community  $k$  with a probability  $p(z_i = k)$ , and  $v_i$ 's "closeness" to community  $k$  is measured by a probability  $p(v_i | z_i = k; \phi, \psi, \Sigma_k)$ . We define  $p(v_i | z_i = k; \phi, \psi)$  as a multivariate normal distribution with the mean  $\psi_k$  and the covariance  $\Sigma_k$ :

$$p(v_i | z_i = k; \phi, \psi) = \mathcal{N}(\phi_i | \psi_k, \Sigma_k). \quad (7)$$

For notation simplicity, we further denote  $\pi_{ik} = p(z_i = k)$ , where  $\pi_{ik} \in [0, 1]$  and  $\sum_{k=1}^K \pi_{ik} = 1$ . Finally, we define the objective function for using community embedding to enforce the higher-order proximity as

$$O_3 = -\frac{\beta}{K} \sum_{i=1}^N \log \sum_{k=1}^K \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k), \quad (8)$$

where  $\beta > 0$  is a trade-off parameter. By minimizing  $O_3$ , we: 1) find the community assignment  $\pi_{ik}$ 's for each node  $v_i$ ; 2) optimize the community embedding ( $\psi_k, \Sigma_k$ )'s to best explain their corresponding community member nodes; 3) achieve the higher-order proximity by making the nodes within the same community to have similar embeddings.

**Joint modeling.** Given all the first-order, second-order and higher-order proximities, we now jointly optimize the node embedding and the community embedding. Denote  $\Phi = [\phi_1, \dots, \phi_N] \in \mathbb{R}^{d \times N}$ ,  $\Phi' = [\phi'_1, \dots, \phi'_N] \in \mathbb{R}^{d \times N}$ ,  $\Psi = [\psi_1, \dots, \psi_K] \in \mathbb{R}^{d \times K}$  and  $\Sigma = \{\Sigma_1, \dots, \Sigma_K\}$ . Then we aim to minimize the overall objective function for ComEmbed:

$$\mathcal{L}(\Phi, \Phi', \Psi, \Sigma) = O_1(\Phi) + O_2(\Phi, \Phi') + O_3(\Phi, \Psi, \Sigma). \quad (9)$$

We make some interesting connections of the ComEmbed objective function with the existing graph embedding methods. Firstly,  $O_2$  alone is the objective function for DeepWalk. Secondly, LINE trains  $O_1$  and  $O_2$  separately, but it also suggests jointly training  $O_1$  and  $O_2$  to combine two proximity. Our  $O_1 + O_2$  can be seen as an extension of LINE. Thirdly, as shown in next section, we solve Eq. 9 by iteratively optimizing node embedding and community embedding. In contrast, first using LINE to learn node embedding and then using GMM to learn community embedding is a pipeline approach (to be evaluated in the community prediction task of our experiment). Finally, as GMM is known as a probabilistic version of K-means clustering (Bishop 2006), Eq. 9 can also be seen a probabilistic and joint optimization version of "first doing LINE then doing K-means".

## Inference

To solve Eq. 9, we do coordinate descent between  $(\Phi, \Phi')$  and  $(\Psi, \Sigma)$ . Given  $(\Phi, \Phi')$ , optimizing  $(\Psi, \Sigma)$  equals to detecting communities in the embedded space. Given  $(\Psi, \Sigma)$ , optimizing  $(\Phi, \Phi')$  equals to learning node embedding with all the first-order, second-order and higher-order proximities. By iteratively optimizing  $(\Phi, \Phi')$  and  $(\Psi, \Sigma)$ , we keep

minimizing the objective function. As the objective function is bounded, we eventually reach the convergence.

**Fixing  $(\Phi, \Phi')$ , optimize  $(\Psi, \Sigma)$ .** In this case, we simplify  $\mathcal{L}(\Phi, \Phi', \Psi, \Sigma)$  as the negative likelihood of a GMM. According to (Bishop 2006), we can easily optimize  $(\Psi, \Sigma)$  by expectation maximization, and fortunately we have closed-form update for each parameter as:

$$\psi_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} \phi_i, \quad (10)$$

$$\Sigma_k = \frac{1}{N_k} \sum_{i=1}^N \gamma_{ik} (\phi_i - \psi_k)(\phi_i - \psi_k)^T, \quad (11)$$

$$\pi_{ik} = \frac{N_k}{N}, \quad (12)$$

where  $\gamma_{ik} = \frac{\pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k)}{\sum_{k'=1}^K \pi_{ik'} \mathcal{N}(\phi_i | \psi_{k'}, \Sigma_{k'})}$  and  $N_k = \sum_{i=1}^N \gamma_{ik}$ .

**Fixing  $(\Psi, \Sigma)$ , optimize  $(\Phi, \Phi')$ .** In this case, we simplify  $\mathcal{L}(\Phi, \Phi', \Psi, \Sigma)$  as optimizing LINE with community embedding regularization. Because of the summation within the logarithm term in  $\mathcal{L}$ , it is inconvenient to compute the gradient of  $\phi_i$ . As a result, we use *variational inference* and minimize an upper bound of  $\mathcal{L}(\Phi, \Psi, \Sigma)$ . We first define

$$O'_3 = -\frac{\beta}{K} \sum_{i=1}^N \sum_{k=1}^K \pi_{ik} \log \mathcal{N}(\phi_i | \psi_k, \Sigma_k). \quad (13)$$

It is easy to prove that

$$O'_3(\Phi, \Psi, \Sigma) \geq O_3(\Phi, \Psi, \Sigma), \quad (14)$$

due to the log-concavity  $\sum_{i=1}^N \log \sum_{k=1}^K \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k) \geq \sum_{i=1}^N \sum_{k=1}^K \log \pi_{ik} \mathcal{N}(\phi_i | \psi_k, \Sigma_k)$ . As a result, we define

$$\mathcal{L}'(\Phi, \Phi') = O_1(\Phi) + O_2(\Phi, \Phi') + O'_3(\Phi, \Psi, \Sigma),$$

and we have  $\mathcal{L}'(\Phi, \Phi') \geq \mathcal{L}(\Phi, \Phi')$ . Finally, we optimize  $\mathcal{L}'(\Phi, \Phi')$  by stochastic gradient descent (SGD). For each  $v_i \in V$ , we have its gradient as

$$\frac{\partial O_1}{\partial \phi_i} = -\sum_{(i,j) \in E} \sigma(-\phi_j^T \phi_i) \phi_j, \quad (15)$$

$$\begin{aligned} \frac{\partial O_2}{\partial \phi_i} = & -\alpha \sum_{u_j \in C(v_i)} \left[ \sigma(-\phi_j'^T \phi_i) \phi_j' \right. \\ & \left. + \sum_{l=1}^m \mathbb{E}_{u_k \sim P_n(u)} [\sigma(\phi_k'^T \phi_i) (-\phi_k')] \right], \end{aligned} \quad (16)$$

$$\frac{\partial O'_3}{\partial \phi_i} = \frac{\beta}{K} \sum_{k=1}^K \pi_{ik} \Sigma_k^{-1} (\phi_i - \psi_k). \quad (17)$$

Thus, we have the gradient for  $\phi_i$  as

$$\frac{\partial \mathcal{L}'}{\partial \phi_i} = \frac{\partial O_1}{\partial \phi_i} + \frac{\partial O_2}{\partial \phi_i} + \frac{\partial O'_3}{\partial \phi_i}. \quad (18)$$

We also compute the gradient for context embedding as

$$\begin{aligned} \frac{\partial O_2}{\partial \phi_j'} = & -\alpha \sum_{v_i \in V} \sum_{u_j \in C(v_i)} \left[ \delta(u_j = v_j) \sigma(-\phi_j'^T \phi_i) \phi_i \right. \\ & \left. + \sum_{l=1}^m \mathbb{E}_{u_k \sim P_n(u)} [\delta(u_k = v_j) \sigma(\phi_k'^T \phi_i) (-\phi_i)] \right], \end{aligned} \quad (19)$$

---

**Algorithm 1** ComEmbed

---

**Require:** graph  $G = (V, E)$ , number of communities  $K$ , number of paths per node  $\gamma$ , walk length  $\ell$ , context window size  $\zeta$ , embedding dimension  $d$ , size of negative context  $m$ , parameters  $\alpha$  and  $\beta$ .

**Ensure:** node embedding  $\Phi$ , context embedding  $\Phi'$ , community embedding  $(\Psi, \Sigma)$ .

- 1: Initialize a path set  $\mathcal{P} = \emptyset$ ;
- 2: **for all**  $v_i \in V$  **do**
- 3:   **for**  $j = 1 : \gamma$  **do**
- 4:     path  $p \leftarrow \text{SamplePath}(G, v_i, \ell)$ ;
- 5:      $V_p \leftarrow \text{GetNodeSubset}(V, p)$ ;
- 6:     **for all**  $v_k$  exists in  $p$  **do**
- 7:        $C_p(v_k) \leftarrow \text{GetContextSet}(p, v_k, \zeta)$ ;
- 8:        $\bar{C}_p(v_k) \leftarrow \text{SampleNegativeContext}(G, v_k, m)$ ;
- 9:        $E_p(v_k) \leftarrow \text{GetEdgeSubset}(E, p, v_k)$ ;
- 10:     **end for**
- 11:      $\mathcal{P} \leftarrow \mathcal{P} \cup p$ ;
- 12:   **end for**
- 13: **end for**
- 14: Initialize  $\Phi, \Phi', \Psi$  and  $\Sigma$ ;
- 15: **for**  $iter = 1 : T_1$  **do**
- 16:   **for**  $subiter = 1 : T_2$  **do**
- 17:     Update  $\psi_k$  by Eq. 10; update  $\Sigma_k$  by Eq. 11; update  $\pi_{ik}$  by Eq. 12;
- 18:   **end for**
- 19: Shuffle the paths in  $\mathcal{P}$ ;
- 20:   **for all** path  $p \in \mathcal{P}$  **do**
- 21:     **for all**  $v_i \in V_p$  **do**
- 22:       Do SGD to optimize  $\phi_i$  based on  $E_p(v_i), C_p(v_i)$  and  $\bar{C}_p(v_i)$  by Eq. 18;
- 23:     **end for**
- 24:     **for all**  $v_j$  exists as a context for  $V_p$  **do**
- 25:       Do SGD to optimize  $\phi'_j$  based on  $V_p, C_p(v_i)$ 's and  $\bar{C}_p(v_i)$ 's by Eq. 20;
- 26:     **end for**
- 27:   **end for**
- 28: **end for**

---

Finally we have the gradient for  $\phi'_j$  as

$$\frac{\partial \mathcal{L}'}{\partial \phi'_j} = \frac{\partial O_2}{\partial \phi'_j}. \quad (20)$$

**Algorithm and complexity.** We summarize our ComEmbed in Alg. 1. In lines 1–13, we sample paths on the graph and then prepare the positive context set  $C_p(v_k)$ , negative context set  $\bar{C}_p(v_k)$  and edge set  $E_p(v_k)$  for each node in these paths. In lines 16–18, we fix  $(\Phi, \Phi')$  and optimize  $(\Psi, \Sigma)$ . In lines 19–27, we fix  $(\Psi, \Sigma)$  and optimize  $(\Phi, \Phi')$ .

We analyze the time complexity for Alg. 1. Lines 1–13 take  $O(\gamma N \ell)$  to sample paths and node context in each path. Line 14 takes  $O(N + K)$  to initialize the parameters. Line 17 takes  $O(NK)$  to compute a GMM. Line 19 takes  $O(\gamma N)$ . Lines 21–23 take  $O(\ell^2 + \ell K)$  to compute the node embedding gradients. Lines 24–26 take  $O(\ell^2)$  to compute the context embedding gradients. Thus, the overall complexity for Alg. 1 is  $O(\gamma N \ell + N + K + T_1(T_2 N K + \gamma N + \ell^2 + \ell K))$ ,

which is linear to  $N$ . Hence Alg. 1 is efficient. Moreover we can easily parallelize Alg. 1 for each path in lines 20–27.

## Experiments

**Data sets.** We use two real-world data sets<sup>3</sup>: *BlogCatalog* and *Flickr*. The BlogCatalog data set has 10,312 nodes, 333,983 edges and 39 node labels. The Flickr data set has 80,513 nodes, 5,899,882 edges and 195 node labels.

**Tasks.** We evaluate ComEmbed in two tasks: 1) community prediction; 2) node classification. In community prediction, our goal is to predict the most likely community assignment for each node. Since our data sets are labeled, we set the number of communities  $K$  as the number of distinct labels in the data set. As an unsupervised task, we use the whole graph for learning embeddings and then predicting communities for each node. In node classification, our goal is to classify each node into one of the multiple labels. We follow (Perozzi, Al-Rfou, and Skiena 2014) to first train the embeddings on the whole graph, then we randomly split 10% (BlogCatalog) and 90% (Flickr) of the nodes as test data, respectively. We use the remaining nodes to train a classifier by LibSVM ( $c = 1$  for all methods) (Chang and Lin 2011). We repeat 10 splits and report the average performance.

**Evaluation.** In community prediction, we use both *conductance* (Kloster and Gleich 2014) and *normalized mutual information* (NMI) (Tian et al. 2014) as the evaluation metrics. Conductance is basically a ratio between the number of edges leaving a community and that within the community. NMI measures the closeness between the predicted communities with ground truth based on the node labels.

In node classification, we use *micro-F1* and *macro-F1* as the evaluation metrics (Perozzi, Al-Rfou, and Skiena 2014). Micro-F1 is the overall F1 w.r.t. all kinds of labels. Macro-F1 is the average of F1 scores w.r.t. each kind of label.

**Baselines.** To show the advantages of community embedding and its joint modeling with node embedding, we compare with several state-of-the-art node embedding methods.

- *DeepWalk* (Perozzi, Al-Rfou, and Skiena 2014): it models the second-order proximity for node embedding.
- *LINE* (Tang et al. 2015): it models both the first-order and second-order proximities for node embedding.
- *node2vec* (Grover and Leskovec 2016): it extends DeepWalk to exploit homophily and structural roles for node embedding. It also models the second-order proximity.

For all the baselines, we use the publicly available codes released by their authors for experiments. We try to compare with all the baselines on both BlogCatalog and Flickr data sets. However, we are unable to produce results for node2vec on Flickr due to unmanageable out-of-memory errors on a machine with 64GB memory. Thus we exclude node2vec from comparison on Flickr.

**Parameters and environment.** For DeepWalk, node2vec and ComEmbed, we set  $\gamma = 10$ ,  $\ell = 80$ ,  $\zeta = 10$ ,  $m = 5$ .

---

<sup>3</sup><http://socialcomputing.asu.edu/pages/datasets>

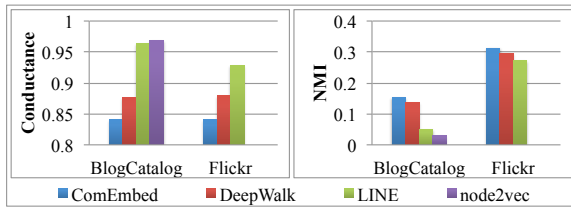


Figure 2: Results on community prediction. The smaller conductance is, the better. The bigger NMI is, the better.

For LINE, we use concatenation of two embeddings for first-order proximity and second-order proximity. In node2vec, we set  $p = 0.25$ ,  $q = 0.25$ . We set the embedding dimension  $d = 128$  for all methods. We always set  $K$  as the number of unique labels in the data sets. We run our experiments on the Linux machines equipped with eight 3.50GHz Intel Xeon(R) CPUs and 16GB memory.

**Task 1. Community Prediction.** We compare ComEmbed with the baselines for community prediction. We set  $\alpha = 1$ ,  $\beta = 0.1$  in ComEmbed for both data sets. As the baselines do not consider community in their embeddings, for fair comparison, we apply GMM over all the methods’ node embedding outputs for community prediction. As shown in Fig. 2, ComEmbed is consistently better than the baselines in terms of both conductance and NMI. Specifically, ComEmbed improves the best baseline (*i.e.*, DeepWalk) in both data sets by relative 4.0%–5.5% (conductance) and 5.3%–11.2% (NMI). This improvement suggests that, for community prediction, modeling community together with node embedding is better than doing them separately.

**Task 2. Node Classification.** We compare ComEmbed with the baselines for node classification. We set  $\alpha = 1$ ,  $\beta = 0.01$  for BlogCatalog and  $\alpha = 0.1$ ,  $\beta = 0.01$  for Flickr. We vary the number of training data to build the classifiers for each method’s node embeddings. As shown in Fig. 3, ComEmbed is generally better than the baselines in terms of both macro-F1 and micro-F1. Specifically, ComEmbed improves the best baselines (*i.e.*, node2vec on BlogCatalog and DeepWalk on Flickr) in both data sets by relative 14.1%–91.8% (macro-F1) and 7.6%–10.2% (micro-F1), when using 90% (BlogCatalog) and 10% (Flickr) of nodes for training. Our student t-tests show that all the above relative improvements are significant over the 10 data splits, with one-tailed  $p$ -values always less than 0.01. It is interesting to see ComEmbed improves the baselines on node classification, since community embedding is after all unsupervised and it does not directly optimize the classification loss. This suggests that the higher-order proximity from community embedding does contribute to a better node embedding.

**Impact of parameters.** We test different values for the model parameters  $\alpha$  and  $\beta$ . As shown in Fig. 4, generally  $\alpha = 0.01$  gives the best results for community prediction and node classification. This suggests keeping an appropriate trade off for the second-order proximity in the objective function is necessary.  $\beta = 0.01$  is generally the best for both tasks. In general, we can see when  $\alpha$  and  $\beta$  are within the

range of  $[0.001, 1]$ , the model performance is quite robust.

## Conclusion

In this paper, we study the problem of embedding communities on the graph. The problem is new because most of the existing graph embedding methods focus on individual nodes, instead of a group of nodes. We observe that community embedding and node embedding reinforce each other. On one hand, a good community embedding helps to get a good node embedding, thanks to its preserving the community structure during embedding. On the other hand, a good node embedding also helps to get a good community embedding, as clustering is then done over the nodes with good representations. We jointly optimize node embedding and community embedding. We evaluate our method on the real-world data sets, and show that it outperforms the state-of-the-art baselines by at least 4.0%–5.5% (conductance) and 5.3%–11.2% (NMI) in community prediction, 14.1%–91.8% (macro-F1) and 7.6%–10.2% (micro-F1) in node classification.

In the future, we plan to include node features in the community embedding. Besides, we also wish to extend our method to handle an infinite number of communities.

## References

- Belkin, M., and Niyogi, P. 2001. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, 585–591.
- Bishop, C. M. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.
- Cao, S.; Lu, W.; and Xu, Q. 2016. Deep neural networks for learning graph representations. In *AAAI*, 1145–1152.
- Chang, C.-C., and Lin, C.-J. 2011. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.* 2(3):27:1–27:27.
- Chang, S.; Han, W.; Tang, J.; Qi, G.; Aggarwal, C. C.; and Huang, T. S. 2015. Heterogeneous network embedding via deep architectures. In *KDD*, 119–128.
- Cox, T. F., and Cox, M. 2000. *Multidimensional Scaling, Second Edition*. Chapman and Hall/CRC, 2 edition.
- Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*.
- Kloster, K., and Gleich, D. F. 2014. Heat kernel based community detection. In *KDD*, 1386–1395.
- Kozdoba, M., and Mannor, S. 2015. Community detection via measure space embedding. In *NIPS*, 2890–2898.
- Levy, O., and Goldberg, Y. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*, 2177–2185.
- Luo, Y.; Wang, Q.; Wang, B.; and Guo, L. 2015. Context-dependent knowledge graph embedding. In *EMNLP*, 1656–1661.
- Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*, 3111–3119.



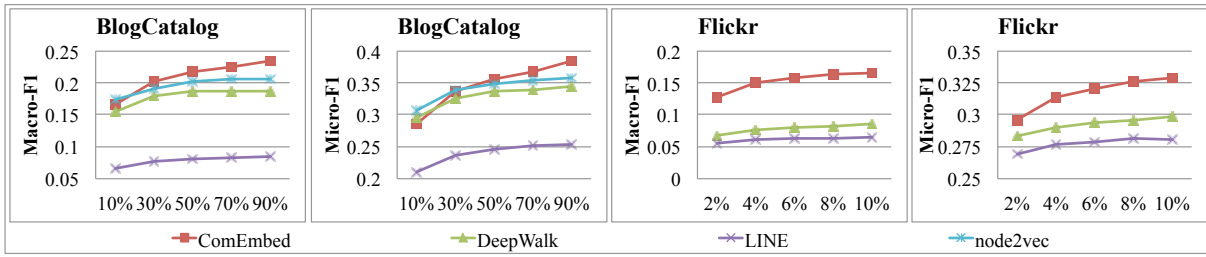


Figure 3: Results on node classification. The bigger macro-F1 and micro-F1 are, the better.

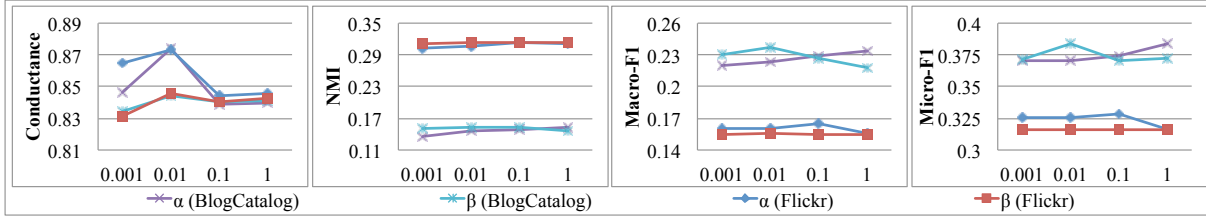


Figure 4: Impact of the parameters  $\alpha$  and  $\beta$ . By default,  $\alpha = 1$ ,  $\beta = 0.01$ .

Niepert, M.; Ahmed, M.; and Kutzkov, K. 2016. Learning convolutional neural networks for graphs. In *ICML*, 2014–2023.

Ou, M.; Cui, P.; Pei, J.; Zhang, Z.; and Zhu, W. 2016. Asymmetric transitivity preserving graph embedding. In *KDD*, 1105–1114.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*, 701–710.

Roweis, S. T., and Saul, L. K. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science* 290(5500):2323–2326.

Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*, 1067–1077.

Tenenbaum, J. B.; de Silva, V.; and Langford, J. C. 2000. A global geometric framework for nonlinear dimensionality reduction. *Science* 290(5500):2319–2323.

Tian, F.; Gao, B.; Cui, Q.; Chen, E.; and Liu, T. 2014. Learning deep representations for graph clustering. In *AAAI*, 1293–1299.

Wang, D.; Cui, P.; and Zhu, W. 2016. Structural deep network embedding. In *KDD*, 1225–1234.

Xie, R.; Liu, Z.; Jia, J.; Luan, H.; and Sun, M. 2016. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, 2659–2665.

Yang, L.; Cao, X.; He, D.; Wang, C.; Wang, X.; and Zhang, W. 2016. Modularity based community detection with deep learning. In *IJCAI*, 2252–2258.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 40–48.